Министерство образования и науки РФ

ФГАОУ ВПО «Уральский федеральный университет
имени первого Президента России Б.Н. Ельцина»

УДК

УТВЕРЖДАЮ
Проректор по науке
_____ Кружаев В.В.
«___» _____ 2013

ОТЧЕТ

О НАУЧНО-ИССЛЕДОВАТЕЛЬСКОЙ РАБОТЕ

В рамках выполнения п.1.2.2.3 Плана реализации мероприятий Программы развития УрФУ на 2013 год

ПО ТЕМЕ:
Разработка алгоритмов трассировки лучей для расчета зонального теплообмена

(Заключительный)

Зав.кафедрой _____ _____
(подпись, дата)

Научный руководитель _____ Лисиенко В.Г.
(подпись, дата)

Исполнитель _____ Коптелов Р.П.
(подпись, дата)

Екатеринбург 2013

Реферат

1. ФИО автора (ов): Коптелов Роман Павлович / Koptelov Roman Pavlovich.

2. Аннотация:
   Целью работы является повышение точности и быстродействия методов трассировки лучей и учета наличия препятствий на пути излучения в системах со сложной геометрией.
   Выполнено сравнительное исследование методов трассировки лучей для расчета зонального теплообмена, а также разработаны программы, реализующие трассировку лучей известными методами. Показано, что стандартный метод трассировки сквозь неравномерную сетку, как наиболее удобную структуру данных для лучистого теплообмена, приводит к проблеме пропадания лучей. Разработан робастный метод трассировки лучей, решающий данную проблему и обладающий немного лучшим быстродействием. Разработан новый алгоритм пересечения прямой с выпуклым многогранником. Алгоритм дает решение проблемы пересечения при неизвестном списке граней многогранника, которая часто возникает в задачах САПР и геометрических вычислениях в реальном времени.
   Все описываемые алгоритмы реализованы в виде программ и готовы к использованию. Полученные результаты могут применяться в моделировании теплообмена, компьютерной графике, САПР и визуализации результатов расчета.

   The aim of this work is increasing accuracy and performance of ray tracing and obstructions handling in systems of complex geometry.
   Known ray tracing methods are compared in context of radiative heat transfer. Methods of ray tracing and radiative view factors evaluation are implemented in computer programs. It is shown that the standard ray tracing with non uniform mesh (the most suitable geometrical structure for radiative heat transfer) fails with loss of some rays in geometry. Robust ray tracing algorithm is developed that solves the problem. The algorithm is also slightly faster than the standard one. New ray – convex polyhedron intersection algorithm is developed. It solves the intersection problem if polyhedron faces are unknown – usual case in CAD and real-time calculations.
   All described algorithms are implemented as computer programs and are ready to use.
   Obtained investigation results can be applied in modeling of radiative heat transfer, computer graphics, CAD and scientific visualization.

3. Ключевые слова: лучистый теплообмен, трассировка лучей, пересечение, многогранник, геометрические структуры данных, разбиение пространства, ограничивающие объемы / radiative heat transfer, ray tracing, intersection, geometrical data structures, space subdivision, bounding volumes

4. Тема отчета: Разработка алгоритмов трассировки лучей для расчета зонального теплообмена. / Ray tracing algorithms for zonal radiative heat transfer

Отчет содержит 28 страниц, 6 таблиц, 19 рисунков и 22 источников литературы.

Содержание

Обозначения и сокращения 3

Введение 4

1. Методы трассировки лучей 6

1.1 Ограничивающие объемы и иерархия ограничивающих объемов 6

1.2 Двоичное разбиение пространства 7

1.3 Равномерная и иерархическая сетка 8

1.4 Использование неравномерной объемной сетки 9

1.5 Перспективы различных методов трассировки лучей для лучистого теплообмена 10

2. Робастный алгоритм трассировки лучей сквозь неравномерную шестигранную сетку 11

2.1 Примеры шестигранной сетки 11

2.2 Трассировка лучей сквозь неравномерную шестигранную сетку: стандартный подход 12

2.3 Робастный метод трассировки без пропадания лучей 14

2.4 Сокращение требуемой памяти 15

3. Пересечение прямой с выпуклым многогранником, представленным списком ребер 17

3.1 Способы представления многогранника 17

3.2 Известные алгоритмы 19

3.2.1 Первые алгоритмы со сложностью (N) 19

3.2.2 Алгоритмы, использующие представление прямой в виде пересечения плоскостей 20

3.3 Разработка алгоритма, использующего список ребер 21

3.3.1 Алгоритм, использующий список ребер, основанный на методе «грубой силы» 21

3.3.2 Алгоритм, использующий список ребер в виде таблицы соседних вершин 23

3.4 Тестирование быстродействия алгоритмов 25

Заключение 27

Список использованных источников 28

Обозначения и сокращения

A – площадь поверхности

D – расстояние от плоскости до начала координат (в нормальном уравнении плоскости)

F_{ij} – угловой коэффициент излучения

i,j,k – инжексы ячеек сетки

M – количество граничных поверхностей

$\overline{n}$, $\overline{N}$ – нормаль к плоскости

N – количество граней многогранника

Nv – количество вершин многогранника

Nx, Ny, Nz – число узлов или ячеек сетки по координатам x, y, z

$\overline{O}$ – начало прямой

r – расстояние между двумя точками

$\overline{R}$ – радиус-вектор

t – расстояние от начала прямой до некоторой точки на прямой

v – видимость между двумя площадками

x,y,z – координаты

θ – угол между нормалью к плоскости и вектором, соединяющим две точки

CAD – система автоматизированного проектирования (САПР)

BB – Bounding Box – ограничивающий параллелепипед

BSP – Binary Space Partitioning – двоичное разбиение пространства

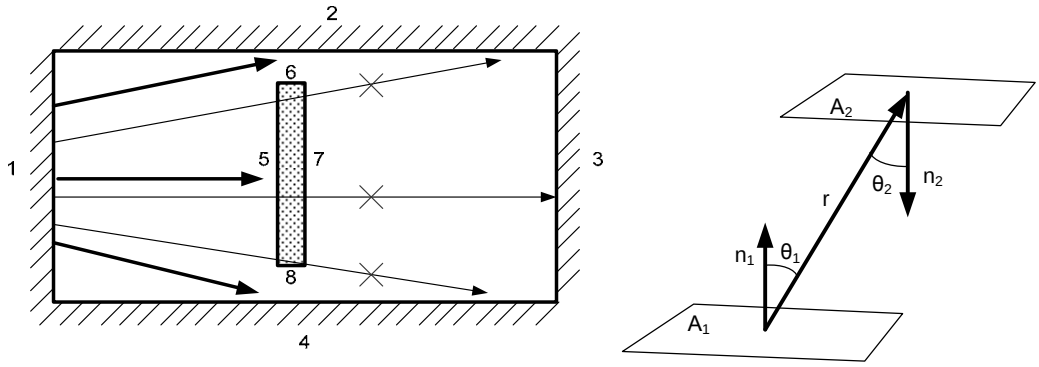
BV – Bounding Volume – ограничивающий объем

BVH – Bounding Volume Hierarchy – иерархия ограничивающих объемов

Введение

Для решения задач радиационного теплообмена наиболее эффективным признан зональный метод расчета, который основан на вычислении угловых коэффициентов излучения для каждой пары поверхностных и объемных зон. Угловой коэффициент излучения для двух зон равен доле энергии излучения, пришедшего с первой зоны и поглощенного второй зоной. Во многих работах, посвященных теории теплообмена, рассматриваются упрощенные модели технических систем: металлургических печей и других высокотемпературных агрегатов, в которых излучение беспрепятственно проходит от одной зоны к другой. В реальности большинство систем имеет сложную геометрию, в которой излучение экранируется, блокируется имеющимися в системе препятствиями: заготовками, стенками самой печи, стенками различных каналов. Наличие препятствий на пути излучения приводит к тому, что уравнения теплового баланса, составленные с помощью угловых коэффициентов, вычисленных по стандартным формулам, дают сильно искаженный результат, так как не учитывают экранирование излучения.

Например, на рис. 1а видно, что большая часть излучения с зоны 1 на зоны 2,3,4 блокируется препятствием и поглощается зоной 5.



а б

Рис. 1. Учет теплового излучения: а) пример системы с экранированием излучения; б) Иллюстрация формулы вычисления углового коэффициента.

В системах с простой геометрией при отсутствии экранов угловые коэффициенты рассчитываются [1] по формуле 1. В отсутствие поглощающей и рассеивающей среды угловой коэффициент с поверхности на поверхность:

$$F_{12} = \frac{1}{\pi \mathrm{A}_1} \int_{A1} \int_{A2} \frac{\cos(\theta_1)\cos(\theta_2)}{\mathrm{r}^2} d\mathrm{A}_2 d\mathrm{A}_1 , \qquad (1)$$

где F_{12} - значение углового коэффициента; A_1 и A_2 - площади поверхностей 1 и 2; θ_1 и θ_2 - углы между нормалями $\bar{\mathrm{n}}_1$ и $\bar{\mathrm{n}}_2$ и центрами элементарных площадок dA_1 и dA_2, r – расстояние между этими элементарными площадками (Рис. 1б).

Однако, стандартная формулы вычисления зонального теплообмена никак не учитывают наличие в системе других поверхностей. Вычисление угловых коэффициентов

без учета препятствий, когда большая часть излучения экранируется, может привести к погрешностям до 100 % при определению тепловых потоков и температур зон.

Для того, чтобы учесть наличие препятствий на пути излучения, для каждой пары элементарных площадок и элементарных объемов проверяется их видимость. Между их центрами проводится луч, который проверяется на пересечение со всеми другими непрозрачными поверхностями. Для численного интегрирования и учета наличия препятствий используется дискретный вариант формулы (1), с умножением на функцию видимости [2]:

$$F_{12} = \frac{1}{\pi A_1} \sum_i \sum_j \frac{\cos(\theta_{1i})\cos(\theta_{2j})}{r_{ij}^{\,2}} v_{ij}\, \Delta A_{2j} \Delta A_{1i}\,, \qquad (2)$$

где i, j – номера элементарных площадок на поверхностях 1 и 2, v_{ij} - видимость площадок ΔA_{1i} и ΔA_{2j}. Если луч, соединяющий центры площадок ΔA_{1i} и ΔA_{2j} не пересекается ни с какими другими поверхностями, то $v_{ij} = 1$, иначе $v_{ij} = 0$.

Однако процесс определения экранирования излучения требует на несколько порядков больше вычислительных затрат, чем вычисление угловых коэффициентов, тепловых потоков и температур в системах с простой геометрией без экранирования.

В больших системах со сложной геометрией количество поверхностных и объемных элементов может достигать тысяч или десятков тысяч, а количество испускаемых лучей миллионов или миллиардов. В принципе, количество лучей не ограничено. Прямой метод трассировки лучей, по сравнению с которым сравнивают другие усовершенствованные методы [3,4], состоит в том, что проверяется пересечение каждого луча со всеми экранами (площадками) за исключением излучающей и принимающей поверхностей. Прямой метод имеет огромную трудоемкость, так как общее количество проверок пересечений зависит от числа поверхностей M как $O(M^3)$. В такой ситуации важнейшую роль играют методы ускорения трассировки лучей, которые не требуют проверки пересечения луча с каждым экраном, см. классическую монографию [5] и современный обзор [6].

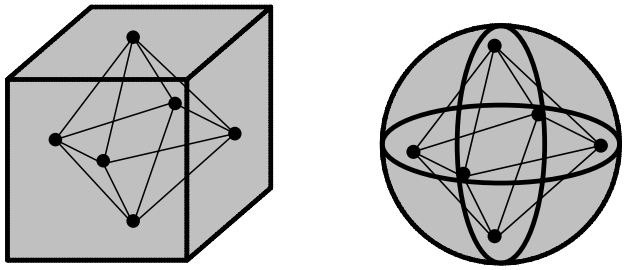
Итак,

1) В большинстве систем имеются препятствия, которые блокируют прохождение излучения;
2) Вычисление угловых коэффициентов без учета экранирования излучения приводит к погрешностям до 100 % при определении температур и тепловых потоков;
3) Учет экранирования излучения требует на несколько порядков больше времени, чем расчет радиационного теплообмена без экранирования.

1. Методы трассировки лучей

1.1 Ограничивающие объемы и иерархия ограничивающих объемов

Применение ограничивающих объемов (Bounding Volume) заключается в следующем: каждый экран заключается в ограничивающий объем простой формы, пересечение луча с которым занимает гораздо меньше времени, чем пересечение с исходным экраном. Пересечение луча с экраном выполняется, только если луч пересекается с ограничивающим объемом. Таким образом, значительно сокращается количество пересечений лучей с экранами [3,5]. В качестве ограничивающих объемов обычно используются прямоугольный параллелепипед (Bounding Box) и сфера [7] (Bounding Sphere) (рис. 2). Грани параллелепипеда обычно выбираются параллельными координатным плоскостям, ориентированные параллелепипеды получили гораздо меньшее распространение.



а б

Рис. 2. Ограничивающие объемы: а – прямоугольный параллелепипед; б – сфера.

Следующим значительным шагом в ускорении трассировки лучей является построение иерархии ограничивающих объемов: для каждой группы ограничивающих объемов строится свой ограничивающий объем «более высокого уровня». Получающаяся структура данных называется иерархией ограничивающих объемов (BVH – Bounding Volume Hierarchy) и хранится в виде двоичного дерева. Пример иерархии для случая параллелепипедов приведено на рис. 3.

Все ограничивающие объемы предварительно сортируются по координате, чтобы каждый из них включал в себя только близлежащие объекты.

Существует два основных подхода к построению иерархии ограничивающих объемов: сверху вниз и снизу вверх [6]. При построении сверху вниз главный ограничивающий объем делится на две части, содержащие примерно одинаковое число объемов более низкого уровня или же на две части, имеющие одинаковый размер, затем процедура применяется к каждому получившемуся объему. Критерием остановки может служить высота дерева или количество объемов внутри, при котором деление нецелесообразно [4]. При построении снизу вверх близлежащие ограничивающие объемы объединяются до тех пор, пока не образуется единственный главный ограничивающий объем.

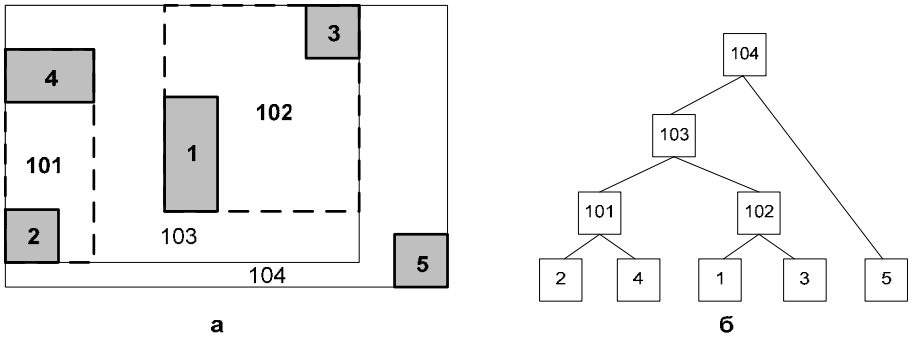


Рис. 3. Иерархия ограничивающих объемов: а – построение; б – хранение в виде двоичного дерева. Параллелепипеды 2,4 и 1,3 заключаются в параллелепипеды более высокого уровня 101 и 102, которые, в свою очередь, заключаются в параллелепипед 103 и т.д.

1.2 Двоичное разбиение пространства

Метод двоичного разбиения пространства также использует ограничивающие объемы, один ограничивающий объем, содержащий все объекты геометрии, и иерархическую структуру в виде дерева, называемого kd-деревом или BSP деревом (BSP – Binary Space Partitioning – двоичное разбиение пространства). Существенные отличия состоят в следующем:

1) BSP дерево всегда строится сверху вниз;
2) Делится на две части не группа объектов (ограничивающих объемов), а пространство;
3) В BSP дереве какой-либо ограничивающий объем может принадлежать одновременно нескольким объемам более высокого уровня (рис. 4). В иерархии ограничивающих объемов каждый объем принадлежит только одному объему более высокого уровня.

Ускорение трассировки лучей достигается за счет способа прохода по дереву и выборочной проверки пересечения луча с параллелепипедами. Существует несколько таких способов [8], однако наиболее распространенным способом, применяемым как для BSP дерева, так и для иерархии ограничивающих объемов, является следующий алгоритм:

1) Проверяется пересечение луча с главным параллелепипедом (ограничивающим объемом). Если пересечения нет, то луч не пересекается ни с одним из экранов.
2) Если луч пересекается с параллелепипедом, то проверяется пересечение с первым дочерним параллелепипедом. Если пересечения нет, то луч должен пересекаться с другим дочерним параллелепипедом. Если пересечение с первым дочерним параллелепипедом есть, то для него определяется список дочерних параллелепипедов, и проверяется пересечение с ними.

3) Шаг 2 повторяется, пока не будет найдена точка пересечения с каким-либо экраном. Если таких точек несколько, то определяется ближайшая к началу луча точка пересечения.

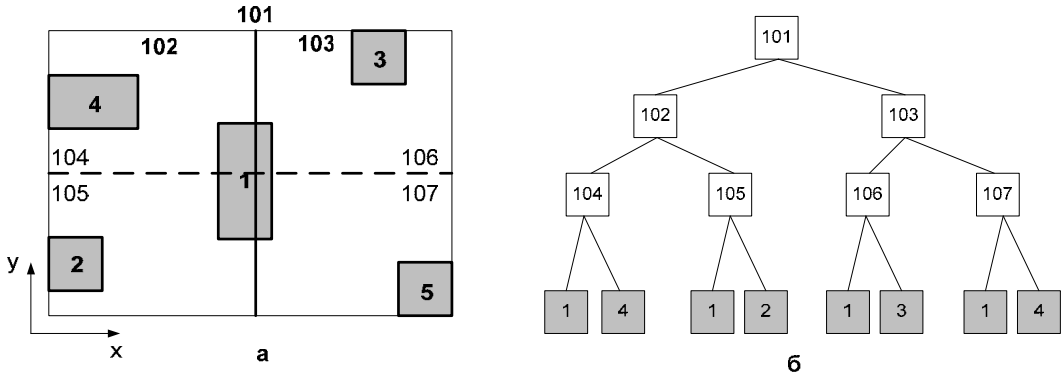


Рис. 4. Двоичное разбиение пространства: а – построение; б – хранение в виде двоичного дерева. Все пространство (расчетная область), заключенное в параллелепипед 101 делится сначала по оси x на параллелепипеды 102 и 103. Затем каждый из получившихся параллелепипедов делится по оси y соответственно на параллелепипеды 104,105 и 106,107. Для каждого из параллелепипедов самого низкого уровня иерархии определяется список содержащихся в нем ограничивающих объемов и объектов. В данном примере ограничивающий объем 1 включен одновременно во все параллелепипеды.

Иерархия ограничивающих объемов и BSP дерево позволяют проверять не все M экранов для каждого луча. Считается, что в лучшем случае достаточно проверить O(log M) экранов, однако на практике этот теоретический предел трудно достижим [4].

1.3 Равномерная и иерархическая сетка

Сетка для трассировки лучей также является методом пространственного разбиения. Вся расчетная область заключается в один параллелепипед, который делится по осям координат на равные промежутки, образуя равномерную сетку. Для каждой ячейки сетки определяется список содержащихся в ней ограничивающих объемов. Каждый ограничивающий объем может принадлежать нескольким соседним ячейкам сетки. Некоторые ячейки сетки могут быть пустыми (рис. 5).

Использование сетки позволяет не проверять пересечение луча со всеми ограничивающими объемами. Существуют различные методы трассировки лучей через сетку [9], которые отличаются способом нахождения следующей по ходу луча ячейки, типом вычислений (целочисленные или с плавающей точкой).

При трассировке лучей последовательно определяются номера ячеек, через которые проходит луч (рис. 5а):

1) Если в текущей ячейке нет объектов, то определяется номер следующей ячейки.
2) Если в текущей ячейке находятся объекты, каждый из них проверяется на пересечение с лучом.

3) Если луч пересекается с одним или несколькими объектами в текущей ячейке, то определяется ближайшая к началу луча точка пересечения и соответствующий объект, на этом трассировка данного луча заканчивается.
4) Если луч не пересекается ни с одним объектом или ячейка сетки пуста, то повторяется процесс по пунктам 1-3.

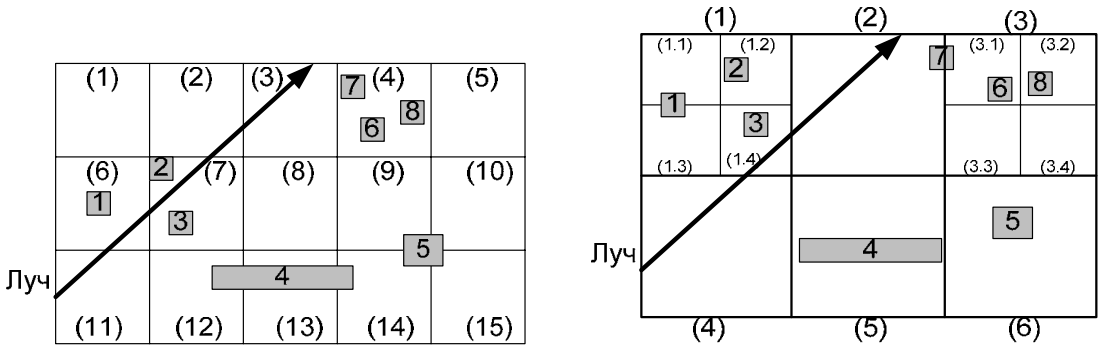


Рис. 5. Использование сетки для трассировки лучей (в скобках указаны номера ячеек.): а) пример равномерной сетки; б) пример иерархической сетки. Ячейки (1) и (3) содержат большое число экранов, поэтому в них построена сетка с меньшим размером ячеек.

В примере на рис. 5а луч испускается из ячейки (11). В ней нет объектов. Следующая по ходу луча ячейка (6), поэтому проверяется пересечение с объектом 1. После ячейки (6) следующая по ходу луча ячейка (7), поэтому проверяется пересечение с объектами 2,3. Затем проверяются ячейки (2) и (3). В итоге трассировка одного луча потребовала проверки 5-ти ячеек и пересечения с 3-мя объектами.

Если экраны распределены в расчетной области неравномерно, то при использование равномерной сетки в некоторых ее ячейках может быть слишком много экранов, каждый из которых будет проверен на пересечение с лучом, или же большинство ячеек будет пустыми, и трассировка лучей сквозь длинный ряд пустых ячеек будет бесполезной тратой времени. В этих случаях применяют иерархические сетки [9]. Чтобы построить иерархическую сетку, необходимо построить крупную равномерную сетку, затем выбрать ячейки, в которых содержится наибольшее количество экранов, и внутри этих ячеек построить равномерную сетку с меньшим размером ячеек (рис. 5б).

В приведенном на рис. 5б примере сетка имеет два уровня. При трассировке изображенного луча сначала проверяется ячейка (4), затем ячейка (1). При переходе к ячейке (1) извлекается информация о ячейках второго уровня, и проверяется ячейка (1.4). В конце проверяется ячейка (2).

1.4 Использование неравномерной объемной сетки

Совершенно отдельной структурой является неравномерная объемная сетка, обычно шестигранная или тетраэдральная (рис. 6), которая частично повторяет форму аппроксимируемого объекта, например, пространства печи с заготовками. В последнее время интерес к использованию такой сетки сильно возрос [4], однако методы

трассировки лучей с ее использованием [10-12] мало разработаны по сравнению с методами, использующими другие структуры данных.

Ячейки такой сетки не содержат внутри себя других объектов, и каждая ячейка является пустой (прозрачной для излучения), заполненной газом или является частью твердого тела. Границами твердых тел являются грани ячеек. На каждом шаге трассировки лучей определяется через какую грань текущей ячейки выходит луч. По номеру грани определяется номер следующей ячейки, через которую луч проходит. Если же грань является границей твердого тела (стенкой печи или заготовок), то трассировка луча прекращается.

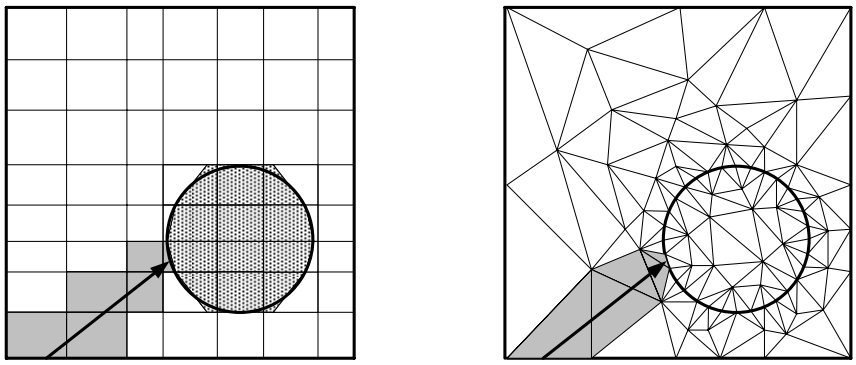


Рис. 6. Пример неравномерной сетки – участок печи с цилиндрической заготовкой (двумерный вариант): а – структурированная четырехугольная сетка; б - неструктурированная треугольная сетка [10]. Серым закрашены ячейки, через которые проходит луч до встречи с экраном – заготовкой.

Преимуществами использования неравномерной объемной сетки как структуры данных для ускорения трассировки лучей являются: хорошее качество аппроксимации геометрии и использование простых алгоритмов пересечения: используются только пересечения с гранями ячеек – четырехугольниками и треугольниками. Недостатки: в отличие от равномерной сетки, нет данных об использовании иерархических конечно-элементных сеток; если большая часть экранов расположена в небольшом объеме геометрии, то большинство ячеек, пересекаемых лучом, являются пустыми, и обработка этих пересечений неэффективна.

1.5 Перспективы различных методов трассировки лучей для лучистого теплообмена

Каждый из описанных методов трассировки лучей имеет свои преимущества и недостатки. Некоторые методы, такие как BSP-дерево, получили наибольшее распространение в связи с широким применением в компьютерной графике, поэтому этот метод наиболее разработан. Использование же конечно-элементной сетки пока недостаточно исследовано, и возможности этого метода, скорее всего, используются не полностью.

Считается, что нельзя выделить один самый лучший метод из описанных выше для любой геометрии. Однако не все эти методы подходят для трассировки лучей в поглощающей и рассеивающей среде. Дело в том, что методы двоичного разбиения пространства, иерархии ограничивающих объемов, равномерной и иерархической сетки ориентированы на вычисление пересечений только с непрозрачными поверхностями в геометрии. При трассировке лучей в поглощающей среде необходимо определить не только поверхность, с которой столкнулся луч, но и все прозрачные грани сеточной модели, через которые луч прошел – чтобы вычислить длину луча, проходящего через каждую объемную зону модели и определить долю поглощенной энергии в каждой зоне. Для этой задачи перечисленные методы ускорения трассировки лучей не могут быть применены без существенной доработки. В отличие от них, метод использования неравномерной объемной сетки позволяет легко найти весь список граней, через которые прошел луч, и таким образом определить долю энергии, поглощенной в каждой объемной зоне с минимальными дополнительными трудозатратами. В связи с этим, использование неравномерной объемной сетки как структуры данных для ускорения трассировки лучей в задачах лучистого теплообмена является наиболее перспективным.

2. Робастный алгоритм трассировки лучей сквозь неравномерную шестигранную сетку

2.1 Примеры шестигранной сетки

В данной работе при проведении численных экспериментов использовались три сетки (рис. 7):

1) Равномерная прямоугольная сетка – куб, имеющий по 10 ячеек в каждом направлении. Координаты углов ячеек генерируются по формулам: $x_{ijk} = i;\quad y_{ijk} = j;\quad z_{ijk} = k$.
2) Косоугольная сетка с одинаковым размером ячеек – куб, имеющий по 10 ячеек в каждом направлении. Координаты углов ячеек генерируются по формулам: $x_{ijk} = i + 0.15j + 0.15k;\quad y_{ijk} = j;\quad z_{ijk} = k$. Величина 0.15 взята произвольно и не имеет принципиального значения для исследования.
3) Криволинейная сетка, представляющая собой половину цилиндра с продольным отверстием. Количество ячеек сетки: 14x29x11. Сетка была построена в программе Gambit.

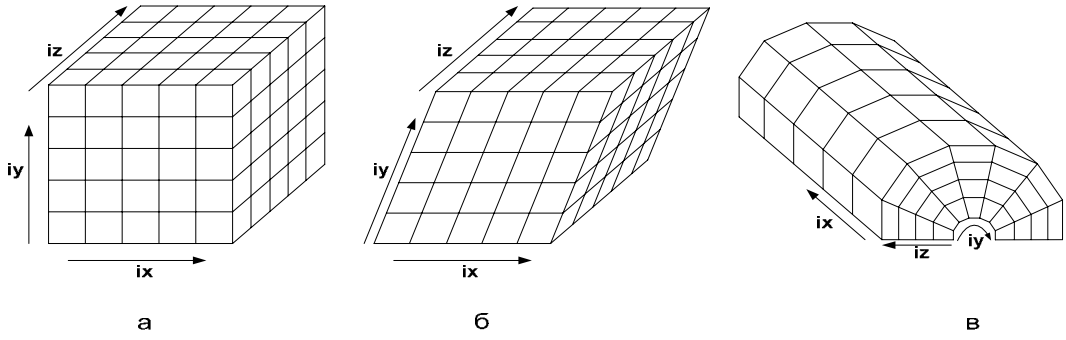


Рис. 7. Примеры шестигранной сетки: а – равномерная прямоугольная сетка; б – косоугольная сетка с одинаковым размером ячеек; в – неравномерная криволинейная сетка. Направления, вдоль которых увеличиваются индексы i,j,k ячеек обозначены соответственно ix, iy, iz.

2.2 Трассировка лучей сквозь неравномерную шестигранную сетку: стандартный подход

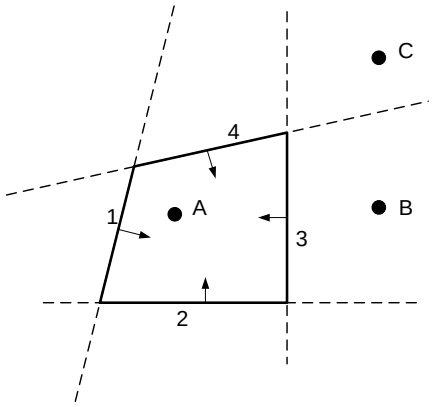
Неравномерная сетка отличается от равномерной не только тем, что имеет переменный размер ячеек вдоль координатных осей, но и тем, что может иметь не прямоугольные ячейки или быть построенной в криволинейных координатах (рис. 7). Очевидный метод трассировки лучей сквозь неравномерную сетку заключается в том, что на каждом шаге проверяется пересечение луча со всеми гранями текущей ячейки, кроме грани, через которую луч вошел. Номер следующей ячейки определяется по тому, с какой гранью пересекается луч [4]. Если пересечение найдено, то оставшиеся грани проверять не нужно.

Для пересечения луча с четырехугольником сначала вычисляется точка пересечения луча с плоскостью, а затем проверяется, принадлежит ли точка четырехугольнику [13].

Существует еще несколько работ в области использования неравномерной сетки для трассировки лучей. В [11] используется тетраэдральная сетка, для трассировки через которую текущий тетраэдр представляется в нормализованных координатах. В [10,12,14] используется тетраэдральная сетка и координаты Плюкера. В [12] используется также и шестигранная сетка, однако каждая четырехугольная грань предварительно разбивается на два треугольника. В отличие от этих работ, здесь используется представление четырехугольника в виде пересечения полуплоскостей.

Описанный метод определения следующей ячейки на пути луча был применен для трех сеток, представленных на рис. 7. Лучи испускались так, что каждый луч соединял центры двух поверхностных граней на границе сетки, за исключением тех случаев, когда грани лежат в одной плоскости. Выяснилось, что при трассировке луча может возникнуть, и даже неоднократно, ситуация, когда луч проходит очень близко или даже точно через ребро сетки между двумя ячейками или через вершину между восемью ячейками. В таких ситуациях, когда проверяется принадлежность точки пересечения граням ячейки, из-за

погрешностей алгоритм не находит ни одной грани, которой принадлежит точка пересечения луча с плоскостью этой грани. То есть, алгоритм сообщает, что луч не выходит ни через одну грань ячейки, и дальше трассировать нельзя. Возникает проблема пропадания лучей. Количество «потерянных» лучей зависит от сетки и показано в табл. 1.



Четырехугольник разбивается на треугольники, и выполняется пересечение луча с треугольником [15], как это сделано в [4] или же четырехугольник представляется как пересечение четырех полуплоскостей. Если точка принадлежит всем четырем полуплоскостям, то она принадлежит четырехугольнику [16] – см. рис. 8.

Рис. 8. Проверка принадлежности точки четырехугольнику. Каждая сторона четырехугольника образует полуплоскость. Точка A принадлежит всем полуплоскостям, точка B – всем, кроме 3-ей, точка C – только 1-ой и 2-ой.

Таблица 1. Результаты стандартного метода трассировки лучей через шестигранную сетку.

Сетка	**Лучей**	**Потерянных лучей**
Равномерная прямоугольная сетка (рис. 9а)	150000	24708 (16.5 %)
Косоугольная сетка (рис. 9б)	150000	22758 (15.2 %)
Неравномерная криволинейная сетка (рис. 9в)	804229	42 (0.0052 %)

Виртуальное расширение граней, при котором точка считается принадлежащей грани, когда она находится внутри нее или хотя бы вблизи ее границ – призвано исключить влияние погрешностей, влияющих на пропадание лучей, однако, как выяснилось, может даже ухудшить ситуацию. Предположим, требуется определить, через какую грань выходит луч из ячейки C1 (рис. 9а). Если сначала будет проверяться грань 2, то, так как луч проходит вблизи нее, алгоритм может посчитать, что луч проходит через грань 2 в ячейку C2, выход из которой может быть уже не найден, даже с учетом принятой допустимой погрешности. Другая ситуация представлена на рис. 9б: известно, что луч прошел через грань 3 в ячейку C3. Начинается проверка пересечения луча с гранями 4,5,6. Если сначала будет проверена грань 4, то с учетом принятой допустимой погрешности, алгоритм может посчитать, что луч выходит из ячейки через грань 4. Можно сделать вывод, что введение допустимой погрешности и расширение граней, чтобы луч точно прошел через одну из них, не решает проблему пропадания лучей.

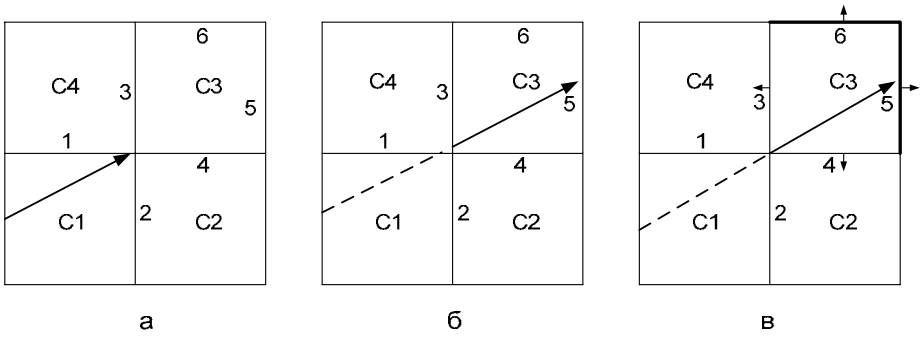


Рис. 9. Иллюстрация различных ситуация при пропадании лучей.

2.3 Робастный метод трассировки без пропадания лучей

Общепринятым методом пересечения прямой с плоскостью [13] является следующий алгоритм:

1) Пусть имеется векторное уравнение прямой $\bar{x}_r = \overline{O} + t\overline{R}$, где $\overline{O}$ - точка на прямой, $\overline{R}$ - вектор направляющих косинусов; и нормальное уравнение плоскости $\overline{N} \cdot \bar{x}_p + D = 0$, где $\overline{N}$ - нормаль к плоскости, D - расстояние от плоскости до начала координат.
2) Проверяется, что прямая и плоскость не параллельны: $|\mathrm{z}| = |\overline{\mathrm{R}} \cdot \overline{\mathrm{N}}| > \varepsilon$, где ε - допустимая погрешность.
3) Вычисляется расстояние t от точки $\overline{O}$ до точки пересечения с плоскостью: $t_p = -(D + \overline{O} \cdot \overline{N}) / z$.
4) Вычисляется точка пересечения в декартовых координатах: $\bar{x} = \overline{O} + t_p \overline{R}$.

Основная идея предлагаемого здесь робастного метода трассировки лучей состоит в том, чтобы, если луч не пересекается ни с одной гранью, принудительно считать пересечением ту грань, расстояние до которой от начала луча наименьшее. Однако этого недостаточно. Например, если текущей ячейкой является С3 (см. рис. 9б), то среди граней 4,5,6 наименьшим от начала луча будет расстояние до грани 4. Это расстояние будет даже меньше, чем расстояние от начала луча до предыдущей найденной точки пересечения с гранью 3. Можно предположить, что для решения задачи достаточно найти наименьшее расстояние из тех, что больше, чем расстояние от начала луча до точки пересечения с входной гранью в ячейку. Однако, если луч проходит точно через ребро или вершину между ячейками, то расстояния до граней входной и до выходной грани могут быть или одинаковыми, или, из-за погрешности, расстояние до выходной грани может быть вычислено меньшим, чем до входной грани. Поэтому, описанное предположение неверно.

Здесь предлагается другое решение: необходимо учитывать ориентацию граней ячейки. Пусть нормали ко всем граням для каждой ячейки ориентированы наружу (рис.

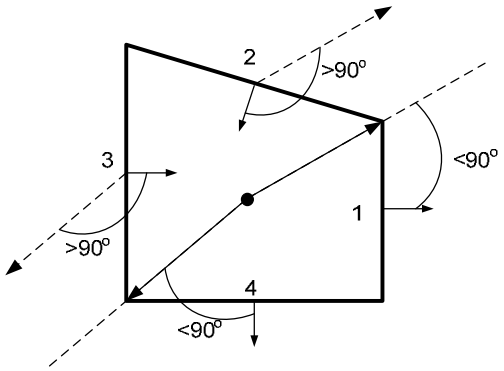
9в). Тогда, чтобы определить грань, через которую луч выходит из ячейки, нужно проверить пересечение только с теми гранями, для которых угол между нормалью и лучом является острым (скалярное произведение $\overline{R} \cdot \overline{N}$ больше нуля). Соответственно, если пересечение не найдено, только из этих граней нужно выбрать ту, расстояние до которой минимально.

Предложенный робастный метод определения следующей ячейки на пути луча был применен для трех сеток, представленных на рис. 7. Результаты эксперимента подтвердили, что все лучи трассируются правильно.

2.4 Сокращение требуемой памяти

Нормали ко всем граням для каждой ячейки вычисляются предварительно для ускорения трассировки лучей. Однако для того, чтобы нормали к граням для каждой ячейки были всегда ориентированы наружу необходимо хранить в памяти $6 \cdot N_c$ векторов, где N_c – количество ячеек в сетке, вместо N_f векторов, где N_f – количество граней в сетке. Например, сетка на рис. 9а имеет 3300 граней (считая внутренние) и 1000 ячеек, тогда $6 \cdot N_c$ = 6000, то есть алгоритм требует почти в два раза больше памяти, чем необходимо. В реальных приложениях размер сетки может быть очень большим, и потому сокращение памяти является важной задачей.

Первый способ сократить количество используемой памяти – хранить лишь N_f нормалей, а для каждой ячейки хранить не нормали к ее граням, а 6 значений «0» или «1», которые определяют, куда направлена нормаль – соответственно внутрь или наружу. Эти значения могут быть вычислены предварительно.



Нормаль направлена наружу ячейки, если угол между нормалью и вектором, идущим от центра ячейки до какой-либо точки, принадлежащей этой грани, например, вершины, является острым (рис. 10).

Рис. 10. Определение ориентации нормалей к граням.

В процессе предварительного вычисления нормалей для всех граней каждая нормаль всегда направлена внутрь одной ячейки и наружу второй, смежной ячейки. Если умножить вектор $\overline{N}$ на «-1», то нормаль будет перенаправлена внутрь второй ячейки и наружу первой. Таким образом, все нормали могут быть принудительно ориентированы так, чтобы все ячейки были разбиты на две группы: для ячеек из первой группы нормали ко всем граням направлены внутрь, а для ячеек второй группы – наружу (рис. 11). Это

возможно как для прямоугольной (шестигранной) сетки, так и для треугольной (тетраэдральной).

С учетом этого алгоритм пересечения луч с плоскостью будет следующим: вычисляется скалярное произведение $z = \overline{R} \cdot \overline{N}$. Если $|z| < \varepsilon$, то луч параллелен плоскости. Если $z > \varepsilon$ и известно, что нормаль направлена наружу, или $z < -\varepsilon$ и известно, что нормаль направлена внутрь, то вычисляется расстояние t от точки $\overline{O}$ до точки пересечения с плоскостью: $t_p = -(D + \overline{O} \cdot \overline{N})/z$. В других случаях данная грань пропускается.

Второй способ, позволяющий провести дальнейшее сокращение памяти, состоит в том, чтобы вместо шести значений, указывающих на ориентацию всех нормалей, хранить только одно значение «1», если все нормали направлены наружу, и «0», если все нормали направлены внутрь.

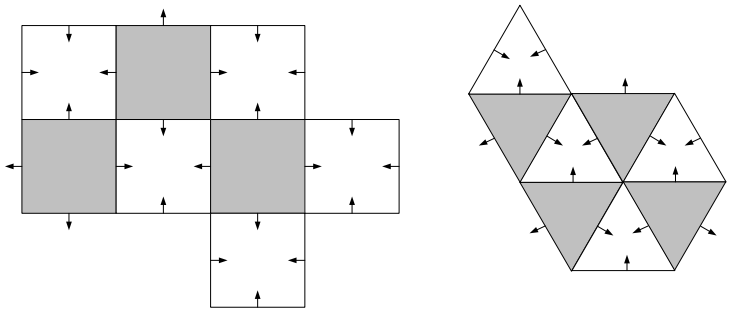


Рис. 11. Принудительно ориентированные нормали к граням. Серым показаны ячейки, для всех граней которых нормали направлены наружу, белым – внутрь; а – пример четырехугольной сетки; б – пример треугольной сетки.

Существует еще один способ сокращение требуемой памяти, он менее универсален, и подходит только к регулярным четырехугольным и шестигранным сеткам. Способ также предполагает, что нормали принудительно ориентированы внутрь или наружу ячеек. Можно заметить, что «серые» и «белые» ячейки чередуются на регулярной четырехугольной сетке, как на шахматной доске (рис. 12). Тогда определить, куда направлены нормали, можно по номеру ячейки в сетке. Например, для сетки на рис. 12 ячейки с четными номерами имеют нормали к граням, направленные наружу, а ячейки с нечетными номерами – внутрь. В этом случае не требуется дополнительной памяти, кроме как для хранения N_f нормалей.

Сравнение по быстродействию двух методов представлено в табл. 2. Стандартный метод показал меньшее время для первых двух сеток за счет того, что он прервал трассировку пропавших лучей, доля которых составляет соответственно 16.5 и 15.2 % (см. табл. 1). Если бы все лучи были обработаны, то время стандартного метода можно оценить соответственно на 16.5 % и на 15.2 % больше, то есть 1.25 и 1.28 секунд, что превышает время робастного метода.

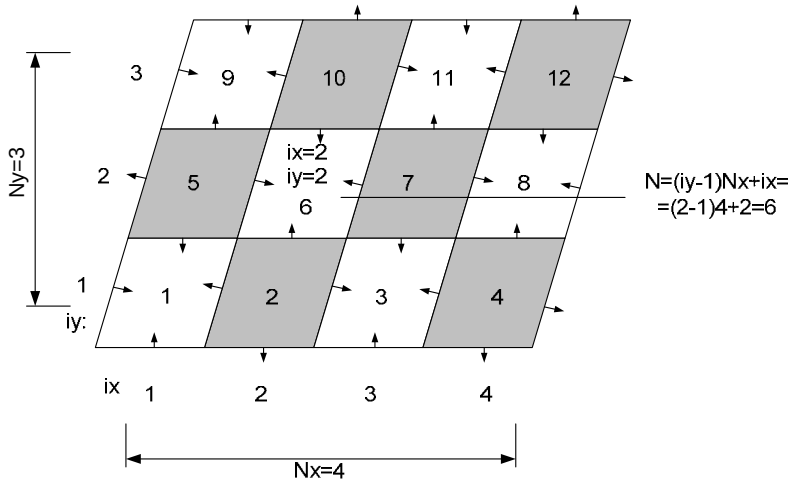


Рис. 12. Определение ориентаций нормалей по номеру ячейки в сетке. Nx, Ny – количество ячеек по осям x,y; ix, iy – индексы ячеек; N – номер ячейки.

Таблица 2. Время трассировки лучей сквозь шестигранную сетку.

Сетка	**Время трассировки, сек**	
	Стандартный метод	**Робастный метод**
Равномерная прямоугольная сетка (рис. 5а)	1.0741	1.1828
Косоугольная сетка (рис. 5б)	1.1089	1.18
Неравномерная криволинейная сетка (рис. 5в)	9.87	9.32

3. Пересечение прямой с выпуклым многогранником, представленным списком ребер

3.1 Способы представления многогранника

Пересечение луча с многогранником является центральной задачей при трассировке лучей, создании изображений, а также при моделировании экранирования теплового излучения. Существует множество алгоритмов решения этой задачи, но они все используют представление многогранника в виде набора граней – многоугольников или треугольников, то есть используют список граней.

Список граней – это таблица, содержащая номера вершин для каждой грани. Таблица хранится в двумерном массиве; в ячейке с индексами (i,j) хранится номер j-ой вершины для i-ой грани. Для многогранника на рис. 13 список граней представлен в табл. 3.

Список ребер может хранится несколькими способами. Первый – в таблице, содержащей номера двух вершин для каждого ребра (табл. 4). Второй – в таблице размером *Nv* x *Nv*, где *Nv* – количество вершин многогранника. Ячейка с индексами (*i*,*j*) содержит 1, если вершины с номерами *i* и *j* соединены ребром, иначе содержит 0. Будем

называть такую структуру таблицей соединений (см. табл. 5). Третий – в таблице размером *Nv* x *Nv*, где *i*-ая строка содержит список соседних вершин для *i*-ой вершины, а последняя позиция в строке содержит количество соседних вершин для *i*-ой вершины. Будем называть такую структуру данных таблицей соседних вершин (см. табл. 6).

Таблица соединений (*c_table*) может быть легко получена из списка граней (*face_list*) с помощью простого алгоритма, псевдокод которого дан ниже (для случая треугольных граней):

```
c_table = 0
for i=1,Nfaces – количество граней
v1 = face_list (i,1)
v2 = face_list (i,2)
v3 = face_list (i,3)
c_table (v1,v2) = 1;  c_table (v2,v1) = 1;
c_table (v1,v3) = 1;  c_table (v3,v1) = 1;
c_table (v2,v3) = 1; c_table (v3,v2) = 1;
end;
```

Таблица вершин (*v_table*) может быть легко получена из таблицы соединений:

```
c_table = 0
for i=1,Nv – количество вершин
for j=1,Nv
if c_table (i,j) = 1 then
v_table (i,nv) = v_table (i,nv )+1  // кол-во соседних вершин
v_table (i, v_table (i,nv)) = j
endif
end
end
```

Таблица 3 Список граней многогранника

Грань\Вершина	V	V	V
F1	1	2	3
F2	1	2	4
F3	1	3	5
F4	1	4	5
F5	2	3	4
F6	3	4	5

Таблица 4 Список ребер многогранника

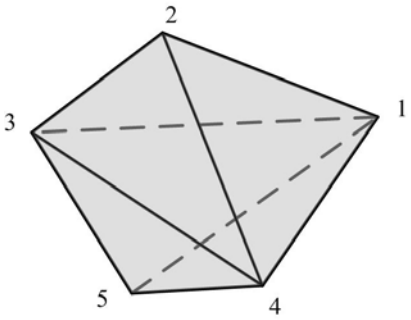
Ребро\Вершина	V	V
E1	1	2
E2	1	3
E3	1	4
E4	1	5
E5	2	3
E6	2	4
E7	3	4
E8	3	5
E9	4	5

Таблица 5 Таблица соединений

Вершина\Вершина	V1	V2	V3	V4	V5
V1	0	1	1	1	1
V2	1	0	1	1	0
V3	1	1	0	1	1
V4	1	1	1	0	1
V5	1	0	1	1	0

Таблица 6 Таблица соседних вершин

Вершина\Вершина	V	V	V	V	Число соединих вершин
V1	2	3	4	5	4
V2	1	3	4		3
V3	1	2	4	5	4
V4	1	2	3	5	4
V5	1	3	4		3



Однако, если имеется только список ребер, то список граней не может быть получен так просто. Например, если вычислять список граней из предположения, что каждый замкнутый цикл вершин или ребер образует грань многогранника, то будут получены ложные грани. Например, для многогранника на рис. 13 будет найдена треугольная грань с вершинами 1,3,4 – однако это будет неверно.

Рис. 13. Пример многогранника.

Итак, список граней не может быть получен простым способом, и поэтому нельзя использовать известные алгоритмы пересечения прямой с многогранником. Предлагаемый здесь алгоритм, использующий список ребер, дает решение этой проблемы.

3.2 Известные алгоритмы

3.2.1 Первые алгоритмы со сложностью (N)

Существует два общих алгоритма, оба из них имеют вычислительную сложность пропорциональную количеству граней - *O(N): алгоритм Cyrus-Beck'a и прямой алгоритм*. Оба этих алгоритма очень популярны и в настоящее время. В прямом алгоритме [17] производится пересечение прямой с каждой гранью многогранника, пока не будет найдено два пересечения или не закончатся грани. В качестве входных данных алгоритм использует координаты точки на прямой **O**, вектор прямой **D**, список *Nv* координат вершин многогранника $\mathbf{P_i}$, количество *Nv* и список граней. Выходными данными являются расстояния от точки **O** до двух точек пересечения t_1,t_2, и координаты точек пересечения $\mathbf{Point_1}$ and $\mathbf{Point_2}$.

Алгоритм *Cyrus-Beck'a* [18] использует представление многогранника в виде пересечения полупространств. Границами этих полупространств являются плоскости, в которых лежат грани многогранника. Грани можно разделить на две группы по их ориентации относительно прямой. Для граней ориентированных лицевой стороной к прямой, на которой задано направление, ищется точка пересечения с максимальным значением параметра t_l. Для граней из другой группы ищется точка пересечения с минимальным значением параметра t_2. Пересечение прямой с многогранником существует, если $t_l \le t_2$ [18]. Этот процесс проиллюстрирован на рис. 14 для двумерного случая.

В качестве входных данных алгоритм *Cyrus-Beck'a* использует координаты точки на прямой **O**, вектор прямой **D**, нормальные уравнения граней $\mathbf{N_i}\cdot\mathbf{P_i} + d_i = 0$, где $\mathbf{N_i}$ - нормальные векторы граней, и количество граней N. Выходными данными являются расстояния от точки **O** до двух точек пересечения t_1,t_2, и координаты точек пересечения $\mathbf{Point_1}$ and $\mathbf{Point_2}$. Значения $\mathbf{N_i}$ и d_i могут быть вычислены предварительно один раз и использоваться во всех пересечениях с множеством прямых.

Полупространства, используемые в алгоритме, образованы гранями многогранника, поэтому алгоритму *необходим список граней*.

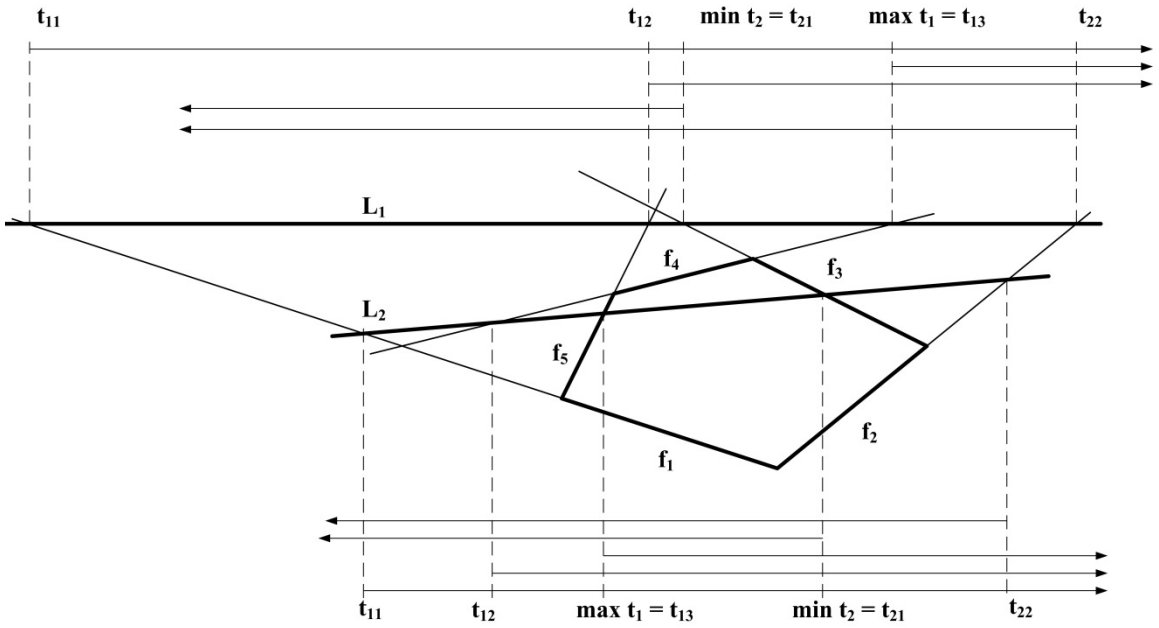


Рис. 14. Нахождение интервалов значений параметра t, при которых прямая пересекает многогрнник, заданный гранями f_1-f_5. Прямая L_1 не пересекает многогранник, потому что $\max(t_1)>\min(t_2)$. Прямая L_2 пересекает многогранник, так как $\max(t_1)<\min(t_2)$.

3.2.2 Алгоритмы, использующие представление прямой в виде пересечения плоскостей

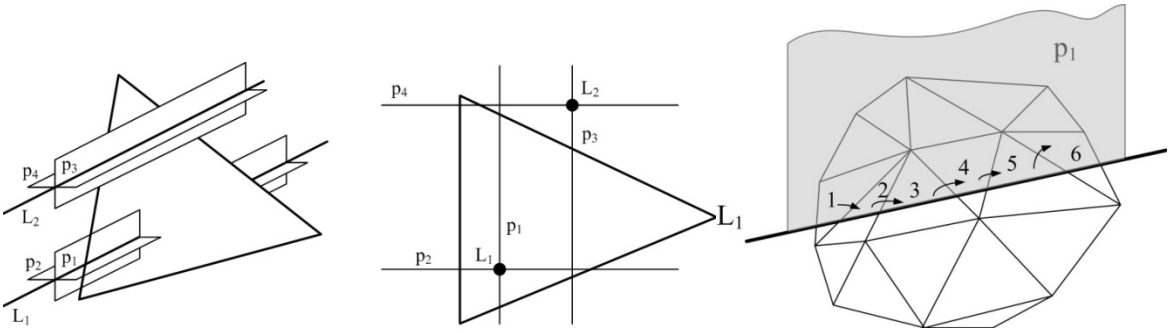
Основная идея ускорения двух первых алгоритмов состоит в исключении части граней из рассмотрения до основных вычислений. Один из возможных путей здесь – проверять наличие пересечения прямой и ограничивающего объема (Bounding Volume, Bounding Box). Такая стратегия обычно применяется для целого многогранника, но не применяется к отдельным граням многогранника. Другая идея была предложена V. Skala [19] для наиболее частого случая треугольных граней. Прямая L_1 представляется пересечением двух плоскостей p_1 и p_2. Если прямая L_1 пересекает некоторый треугольник, то обе плоскости p_1 и p_2 также пересекают этот треугольник. Но если плоскости пересекают треугольник, то прямая может пересекать (прямая L_1 и плоскости p_1 и p_2) или не пересекать (прямая L_2 и плоскости p_3 и p_4), см. рис. 15. Тогда каждая треугольная грань многогранника сначала тестируется на пересечение с плоскостями p_1 и p_2 перед детальным тестом на пересечение прямой с треугольником.

Плоскость пересекается с треугольником тогда и только тогда, когда существуют две вершины треугольника $\mathbf{x_j}$ and $\mathbf{x_k}$, такие что $sign(F_i(x_j)) \neq sign(F_i(x_k))$, где $F_i = 0$ - уравнение i-ой плоскости p_i, $i=1,2$.

Такой предварительный тест может применяться совместно с прямым алгоритмом или с алгоритмом Cyrus-Beck'a. Быстродействие повышается в 2-4 раза для многогранника со 100 гранями и в 3.4 – 6.1 раз для многогранника с 1000 гранями [17].

Такой предварительный тест позволяет определить ребро треугольника, пересеченное плоскостью p_1 или p_2, и следующий треугольник, которому также

принадлежит это ребро. Тогда можно проверять не все грани на пересечение с плоскостями p_1 и p_2, а только «кольцо» треугольников, пересеченное плоскостью p_1. Следуя от одного треугольника к другому (см. рис. 15в), в среднем нужно будет проверить лишь $O(\sqrt{N})$ треугольников [20].



а б в

Рис. 15. Использование двух пересекающихся плоскостей для представления линии: а) общий вид; б) вид по нормали к плоскости треугольника; в) проход последовательности треугольников с общими ребрами.

3.3 Разработка алгоритма, использующего список ребер

3.3.1 Алгоритм, использующий список ребер, основанный на методе «грубой силы»

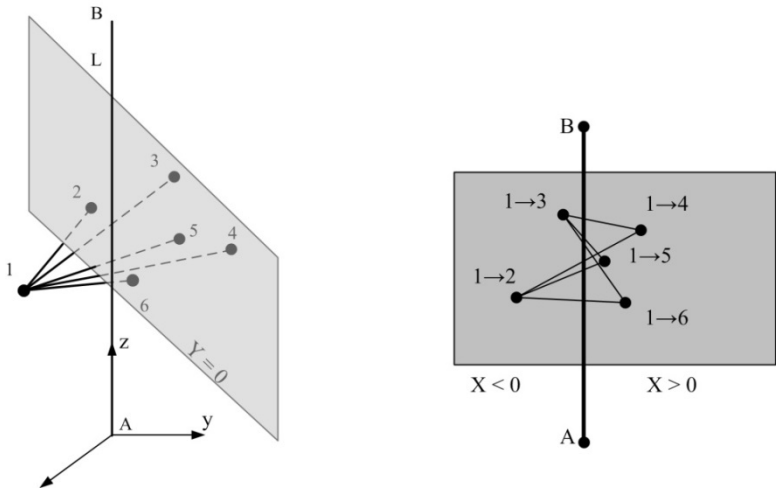
Идея алгоритма появилась из цели использовать наиболее простую структуру данных для хранения многогранника, которая позволяла бы хранить его вершины в произвольном порядке. Алгоритм был предложен в [3] для определения наличия препятствий на пути теплового излучения в металлургической печи. Основная идея алгоритма: построить сечение многогранника плоскостью, содержащей прямую, с которой ищется пересечение, и затем решать задачу пересечения прямой с многоугольником – сечением многогранника.

Алгоритм:

1. Рассмотрим две точки *A* и *B* на прямой *L*. Используем преобразование координат, чтобы поместить точку *A* в начало координат, а точку *B* на ось *z*. На практике любая из осей может быть использована.
2. Вершины многогранника делятся на две группы: $V_{y<0}$ - вершины с у координатой меньше нуля, и $V_{y\geq 0}$ - вершины с у координатой не меньше нуля.
3. Если одна из двух групп не содержит ни одной вершины, то прямая *L* не пересекает многогранник. Иначе переходим на следующий шаг.
4. Каждая вершина $v_i \in V_{y<0}$ соединяется с каждой вершиной из другой группы $v_j \in V_{y\geq 0}$ и вычисляются $N_{y<0} \cdot N_{y\geq 0}$ точек $P_{y=0}$, которые принадлежат сечению многогранника плоскостью *y = 0* (рис. 16а).
5. Точки $P_{y=0}$ делятся на две группы: $P_{x<0}$ - вершины с координатой x меньше нуля, и $P_{x\geq 0}$ - вершины с координатой x не меньше нуля (рис. 16б).

6. Если одна из двух групп не содержит ни одной вершины, то прямая L не пересекает многогранник. Иначе прямая пересекает многогранник, ищем точки пересечения.
7. Каждая вершина $P_i \in P_{x<0}$ соединяется с каждой вершиной из другой группы $P_j \in P_{x\geq 0}$ и вычисляются $N_{x<0} \cdot N_{x\geq 0}$ точек $P_{x,y=0}$, лежащих на прямой L и имеющих только z координату.
8. Находится минимальное и максимальное значение z точек $P_{x,y=0}$. Здесь z имеет то же значение, что и параметр t в алгоритмах, использующих параметрическое уравнение прямой $L = O + t \cdot D$. Теперь можно вычислить точки пересечения: $Point_1 = A + \min z \cdot (B - A)$, $Point_2 = A + \max z \cdot (B - A)$.

Для работы этого алгоритма не требуется ни список граней, ни список ребер. Не используется никакой информации о гранях и ребрах многогранника, - только координаты его вершин, перечисленных в любом порядке.



а б

Рис. 16. Иллюстрация алгоритма, использующего список ребер, основанного на методе грубой силы: а – шаги 2-4; б – шаги 5-7.

Замечания:

1. Множество точек $P_{y=0}$, полученное на шаге 4, не является выпуклым в общем случае, поэтому известные алгоритмы пересечения прямой с многоугольником, например, такие как [21,22], не могут использоваться.
2. Если исходный многогранник не является выпуклым, описанный алгоритм найдет верные точки пересечения прямой с выпуклой оболочкой этого многогранника без построения выпуклой оболочки. Это проиллюстрировано на рис. 17.
3. В наихудшем случае множество вершин V будет разделено плоскостью y = 0 на равные части: $N_{y<0} = N_{y\geq 0} = Nv/2$, где Nv - количество вершин. $P_{y=0}$ также будет разделено плоскостью x = 0 на равные части: $N_{x<0} = N_{x\geq 0} = \left(N_{y<0} N_{y\geq 0}\right)/2 = \left(Nv^2\right)/8$. Полное число соединений вершин между собой будет равно $N_{x<0} \cdot N_{x\geq 0} = \left(Nv^4\right)/64$,

следовательно, в наихудшем случае сложность алгоритма $O(Nv^4)$. В наилучшем случае деление вершин на плоскости осуществится следующим образом: $N_{y<0}=1$, $N_{y\geq 0}=N-1$ и $N_{x<0}=1$, $N_{x\geq 0}=(N-1)-1$. Следовательно, в наилучшем случае сложность алгоритма $O(Nv)$.

Этот алгоритм подходит не только для вычислений в реальном времени, когда список граней и список ребер неизвестны, но также и для приложений САПР. Например, в процессе работы можно выбрать на экране некоторое множество точек, не задавая связи между вершинами, и отправить это множество в программу вычисления пересечений, тогда как ручное задание списка граней или даже списка ребер занимает много времени и попросту скучно – эта функция не должна выполняться человеком. В частности, такой подход был использован в [3] для выделения из конечно-разностной сетки непрозрачных поверхностей стальных заготовок для дальнейшей трассировки лучей между заготовками.

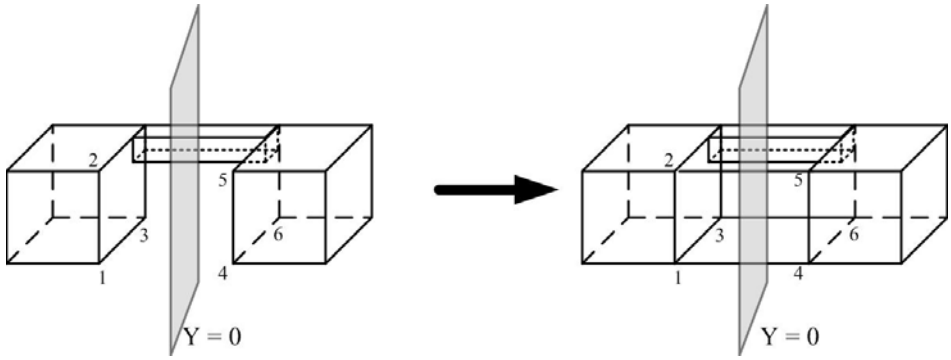


Рис. 17. Иллюстрация работы алгоритма, основанного на методе грубой силы, с невыпуклым множеством точек. Исходное множество точек – 2 куба, соединенных параллелепипедом. Если соединить каждую точку $v_i \in V_{y<0}$ с каждой $v_j \in V_{y\geq 0}$, то вершины 1 и 4, 2 и 5, 3 и 6 будут соединены, и в дальнейших вычислениях будет использована выпуклая оболочка множества V.

3.3.2 Алгоритм, использующий список ребер в виде таблицы соседних вершин

Если для выпуклого многогранника известны не только координаты вершин, но также и список ребер, представленный таблицей соседних вершин (см. введение), то может быть предложен более эффективный алгоритм, чем основанный на методе «грубой силы». Предлагаемый здесь алгоритм, имеет два отличия: не используется преобразование координат, и вершины соединяются между собой не «каждая с каждой», а только в соответствии с таблицей соседних вершин.

Алгоритм:

1. Для прямой L вычисляются уравнения двух плоскостей, пересечением которой является эта прямая: $F_1(x,y,z)=0$ и $F_2(x,y,z)=0$. Плоскости всегда можно выбрать так, чтобы один из коэффициентов A,B или C в уравнении плоскости $Ax+By+Cz+D=0$ был равен нулю, как это было сделано в [19].
2. Вычисляется расстояние D_1 от всех вершин v до плоскости F_1.

3. Вершины делятся на две группы: $V_{D1<0}$, и $V_{D1\geq 0}$ (рис. 18а). Если в одной из групп нет вершин, то прямая не пересекает многогранник, иначе переходим на следующий шаг.
4. Для каждой вершины $v_i \in V_{D1<0}$ имеется список соседних вершин, соединенных с ней ребрами, которые пересекают плоскость $D_1 = 0$. Вычисляются точки пересечения $P_{D1=0}$ этих ребер с плоскостью $D_1 = 0$. Самый эффективный способ сделать это – решить задачу о делении отрезка в заданном отношении: $P_{D1=0} = \frac{v_i \left|D_1(v_j)\right| + v_j \left|D_1(v_i)\right|}{\left|D_1(v_i)\right| + \left|D_1(v_j)\right|}$, так как расстояния $D_1(v_j)$ и $D_1(v_i)$ от вершин до плоскости $D_1 = 0$ уже вычислены. Зная, что $D_1(v_i) < 0$ и $D_1(v_j) \geq 0$, получаем: $P_{D1=0} = \frac{v_i D_1(v_j) - v_j D_1(v_i)}{D_1(v_j) - D_1(v_i)}$. Набор точек $P_{D1=0}$ представляет собой неупорядоченные вершины выпуклого многоугольника.
5. Вычисляется расстояние D_2 от всех точек до плоскости $F_2 = 0$.
6. Точки $P_{D1=0}$ делятся на две группы: $P_{D2<0}$ и $P_{D2\geq 0}$ (рис. 18б). Прямая пересекает многогранник, если в обеих группах есть хотя бы одна точка.
7. Находятся две точки $\mathrm{P}_{\min D2}$ и $\mathrm{P}_{\max D2}$ с минимальным и максимальным значением D_2 и вычисляется параметрическое уравнение прямой, лежащей в плоскости $D_1 = 0$ и проходящей через $\mathrm{P}_{\min D2}$ и $\mathrm{P}_{\max D2}$. Точки $P_{D1=0}$ делятся на четыре группы: $\mathrm{P}_{D2<0,D3\leq 0}$, $\mathrm{P}_{D2<0,D3\geq 0}$, $\mathrm{P}_{D2>0,D3\leq 0}$, $\mathrm{P}_{D2>0,D3\geq 0}$ (рис. 18б). Крайняя точка $\mathrm{P}_{\min D2}$ включена в обе группы $\mathrm{P}_{D2<0,D3\leq 0}$ и $\mathrm{P}_{D2<0,D3\geq 0}$, а $\mathrm{P}_{\max D2}$ включается в обе группы $\mathrm{P}_{D2>0,D3\leq 0}$, $\mathrm{P}_{D2>0,D3\geq 0}$.
8. Находятся точки $\mathrm{P}_{\text{down left}} \in \mathrm{P}_{D2<0,D3\leq 0}$ с макс. значением D_2 и $\mathrm{P}_{\text{down rightt}} \in \mathrm{P}_{D2\geq 0,D3\leq 0}$ с мин. значением D_2. Эти две точки расположены по разные стороны плоскости $F_2 = 0$. Так как $\mathrm{L} = \mathrm{D}_1 \cap \mathrm{D}_2$, то эти две точки расположены по разные стороны прямой L. Тогда первым пересечением прямой L с многогранником будет пересечение прямой L с отрезком $\mathrm{P}_{\text{down left}}\mathrm{P}_{\text{down rightt}}$.
9. Находятся точки $\mathrm{P}_{\text{up left}} \in \mathrm{P}_{D2<0,D3\geq 0}$ с макс. значением D_2 и $\mathrm{P}_{\text{up rightt}} \in \mathrm{P}_{D2\geq 0,D3\geq 0}$ с мин. значением D_2. Пересечение прямой L с отрезком $\mathrm{P}_{\text{up left}}\mathrm{P}_{\text{up rightt}}$ является одновременно вторым пересечением прямой с многогранником.

Предложенный алгоритм использует только координаты вершин многогранника и список ребер, представленный в виде таблицы соседних вершин. Никакой реконструкции граней из списка ребер не производится. Сложность алгоритма $O(Nv)$.

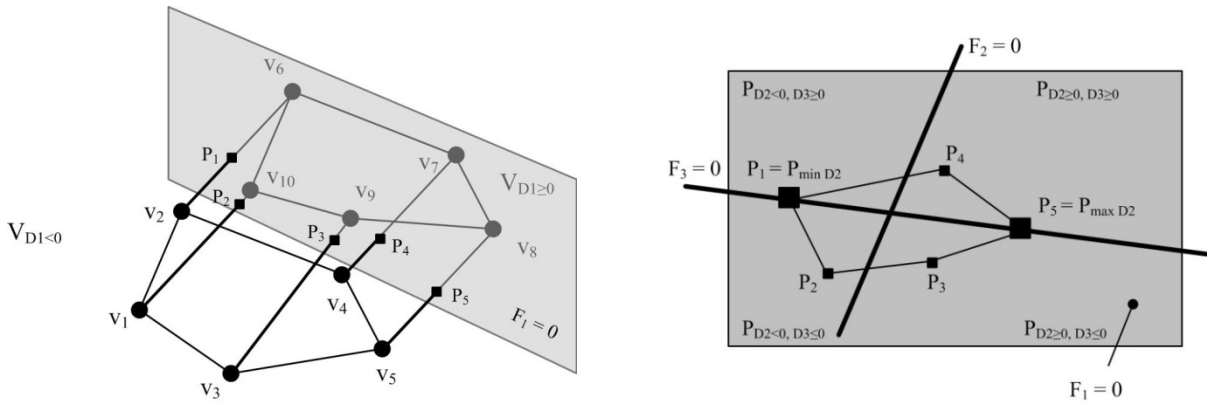


Рис. 18. Иллюстрация предложенного алгоритма: а) Построение сечения многогранника плоскостью $F_1(x)=0$ (шаги 3-4); б) Точки $P_{D1=0}$ делятся на четыре группы: $P_{D2<0,D3\le0}=\{P_1,P_2\}$, $P_{D2<0,D3\ge0}=\{P_1\}$, $P_{D2>0,D3\le0}=\{P_3,P_5\}$, $P_{D2>0,D3\ge0}=\{P_4,P_5\}$. Здесь $P_{\text{down left}}=P_2$, $P_{\text{down right}}=P_3$, $P_{\text{up left}}=P_1$, $P_{\text{up right}}=P_4$. Две точки пересечения прямой с многогранником определяются как $L\cap P_2P_3$ и $L\cap P_1P_4$.

3.4 Тестирование быстродействия алгоритмов

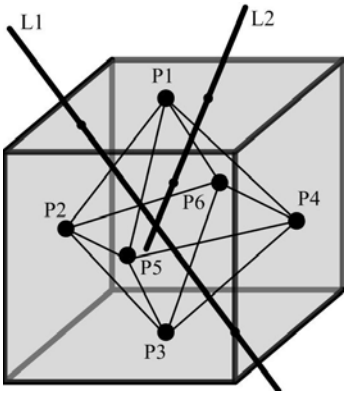
Предложенный алгоритм пересечения прямой с многогранником, использующий список ребер, сравнивался по быстродействию с наиболее известными алгоритмами: прямым и алгоритмом Cyrus-Beck'a. В тестах использовались многогранники с количеством вершин *Nv* от 4 до 120. Для каждого значения *Nv* было сгенерировано 100 выпуклых многогранников со случайным расположением вершин, многогранники были вписаны в куб с центром в начале координат и стороной равной двум. Задача генерирования выпуклых многогранников со случайным расположением вершин нетривиальна. Она решалась следующим образом:

1) В куб помещалось *Nv* точек, сгенерированных с помощью встроенного генератора случайных чисел.
2) Выполнялась программа построения выпуклой оболочки.
3) Если выпуклая оболочка содержит все *Nv* точек, то данное множество точек является выпуклым многогранником. Переходим к генерированию следующего многогранника, пока не наберется нужное количество. Если выпуклая оболочка содержит менее *Nv* точек, возвращаемся на шаг 1.

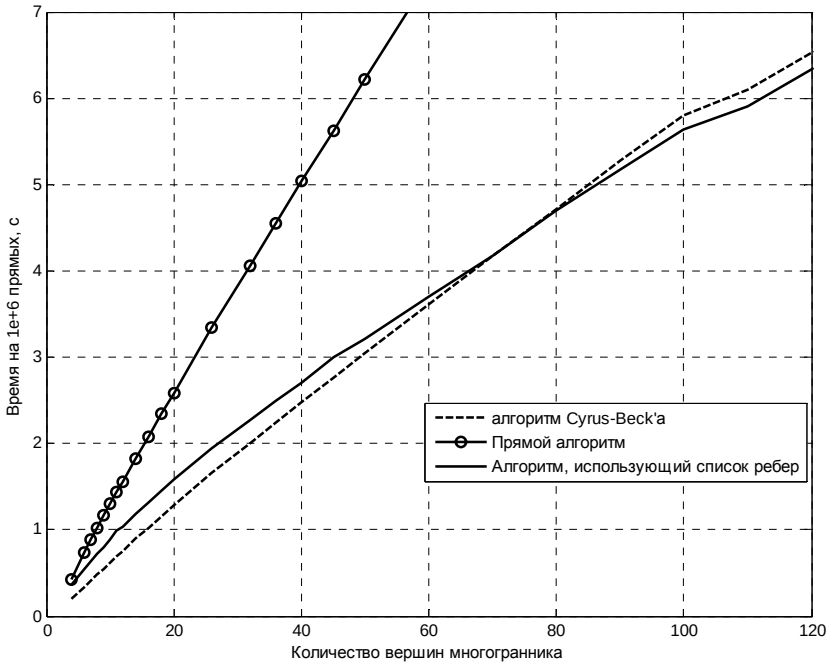
Для каждого многогранника с фиксированным числом вершин для определения пересечения использовалось $2\cdot10^5$ отрезков прямых, то есть всего для каждого значения *Nv* использовалось $2\cdot10^7$ отрезков. Отрезки генерировались случайным образом так, чтобы их концы принадлежали разным граням куба, в который вписаны многогранники (рис. 19а). Такое расположение многогранников и отрезков является наиболее правильным и практичным, потому что, на практике, пересечение прямой с многогранником или другим сложным объектом всегда должно проводиться только после пересечения прямой с ограничивающим объемом (Bounding Box) этого объекта. Для пересечения прямой с треугольником в прямом алгоритме был использован алгоритм, описанный в [15].

На рис. 19б приведено время работы алгоритмов при расчете миллиона пересечений. Все программы исполнялись на процессоре Pentium II 1.83 ГГц.

Видно, что предложенный алгоритм сравнивается по эффективности с наиболее используемым алгоритмом Cyrus-Beck’a, и даже быстрее него при большом количестве вершин многогранника (Nv > 70-80). Это доказывает практическую применимость использования списка ребер вместо списка граней. Если же список граней неизвестен, предложенный алгоритм будет наиболее эффективен, так как список граней не может быть просто получен из списка ребер.



а



б

Рис. 19. Тестирование быстродействия алгоритмов. а) Расположение отрезков прямых и вершин многогранника. Вершины отрезков располагаются на поверхности ограничивающего объема, построенного для многогранника. б) Время расчета миллиона пересечений. Известны и список граней и список ребер.

Заключение

Полученные результаты полностью совпадают с ожидаемыми результатами при планировании НИР:

1) Выполнено сравнительное исследование методов трассировки лучей для расчета зонального теплообмена, а также разработаны программы, реализующие трассировку лучей известными методами. Получен вывод о том, что при трассировке лучей в поглощающей среде необходимо определить не только поверхность, с которой столкнулся луч, но и все прозрачные грани сеточной модели, через которые луч прошел – чтобы вычислить длину луча, проходящего через каждую объемную зону модели и определить долю поглощенной энергии в каждой зоне. Для этой задачи не все перечисленные методы ускорения трассировки лучей могут быть применены без существенной доработки. В отличие от других методов, использование неравномерной объемной сетки позволяет легко найти весь список граней, через которые прошел луч, и определить долю энергии, поглощенной в каждой объемной зоне. В связи с этим, использование неравномерной объемной сетки как структуры данных для ускорения трассировки лучей в задачах лучистого теплообмена является наиболее перспективным.
2) Показано, что стандартный метод трассировки сквозь неравномерную сетку приводит к проблеме пропадания лучей. Разработан робастный метод трассировки лучей, решающий данную проблему и обладающий немного лучшим быстродействием. Метод состоит в том, что: нужно проверить пересечение луча только с теми гранями, для которых угол между нормалью и лучом является острым; если луч не пересекается ни с одной гранью, принудительно считать пересечением ту грань, расстояние до которой от начала луча наименьшее.
3) Предложен новый алгоритм пересечения прямой с выпуклым многогранником. Алгоритм основан на использовании информации о ребрах многогранника (список ребер) и хорошо подходит для случаев, когда нет информации о гранях, такие случаи встречаются в задачах САПР и в геометрических вычислениях в реальном времени. По эффективности алгоритм сравнивается с простыми алгоритмами, использующими список граней, когда он известен, и является даже более эффективным для многогранников с большим количеством вершин или когда список граней неизвестен.

Все описанные в работе алгоритмы реализованы в виде программ и готовы к использованию. По теме НИР были опубликованы 2 работы:

1) Koptelov R.P., Konashkova A.M. Intersection of a Line and a Convex Hull of Points Cloud // Applied Mathematical Sciences, Vol. 7, 2013, no. 103, 5139 – 5149. (скоро появится в SCOPUS, входит в перечень ВАК)
2) «Тепловое излучение: эффективный учет экранов в системах со сложной геометрией» // VIII Всероссийский семинар ВУЗов по теплофизике и энергетике

Еще несколько публикаций в процессе подготовки.

Полученные результаты могут применяться в моделировании лучистого теплообмена, компьютерной графике, САПР и визуализации результатов расчета. Научно-исследовательская работа выполнена успешно.

Список использованных источников

1. Modest, M.F., 2003, Radiative Heat Transfer, Second Edition, Academic Press.
2. Walton, G.N. 2002. “Calculation of obstructed view factors by adaptive integration”, National Institute of Standards and Technology, NISTIR 6925.
3. Г. К. Маликов, В. Г. Лисиенко, Р. П. Коптелов. Методы трассировки лучей для определения угловых коэффициентов излучения в промышленных сложных геометриях. // Известия вузов. Черная металлургия. - 2010. - N 7. - С. 53-59.
4. Mazumder, S., 2008, «Methods to accelerate ray tracing in the Monte Carlo method for surface-to-surface radiation transport». Trans. ASME. J. Heat Transfer, Vol: 128, No: 9.
5. Glassner, A.S., 1989, An introduction to ray tracing, Academic Press.
6. Wald, I., Mark, W.R., Gunther, J., Boulos, S., Ize, T. et al. 2009. “State of the Art in Ray Tracing Animated Scenes”, Computer graphics forum, 28(6), pp. 1691-1722.
7. Whitted, T., 1980, “An improved illumination model for shaded display”. Communications of ACM, 23(6), pp. 343-349.
8. Hapala, M., Havran, V., 2011. “Review: Kd-tree Traversal Algorithms for Ray Tracing”, Computer graphics forum, 30(1), pp. 199-213.
9. Cosenza, B., 2008. “A Survey on Exploiting Grids for Ray Tracing”, Eurographics Italian Chapter Conference.
10. A. Lagae, P. Dutre. 2008. Accelerating ray tracing using constrained tetrahedralizations. Eurographics Symposium on Rendering 27(4), pp. 1303-1312.
11. Favre, J., Lohner, R., 1994. “Ray tracing with a space-filling finite element mesh”, International journal for numerical methods in engineering, 37, pp. 3571-3580.
12. Marmitt, G., Slusallek, P., 2006. “Fast Ray Traversal of Tetrahedral and Hexahedral Meshes”, Eurographics/ IEEE-VGTC Symposium on Visualization.
13. D. Badouel, An Efficient Ray-Polygon Intersection. In Glassner A., editor, Graphics Gems I, pages 390-393. Academic Press, 1990.
14. G. Marmitt, P. Slusallek. 2005. Fast ray traversal of unstructured volume data using Plucker tests. Technical report. Computer Graphics Lab, Saarland University.
15. T. Müller and B. Trumbore, Fast, Minimum Storage Ray-Triangle Intersection, Journal of Graphics Tools, pp. 37-46, (1997), pp.22-28.
16. E. Haines, Point in Poligon Strategies. In Heckbert P.S., editor, Graphics Gems IV, pp. 24-46. Academic Press, 1995.
17. Kolingerova, I.: 3D - Line Clipping Algorithms - A Comparative Study, The Visual Computer, Vol.11, No.2, pp.96-104, 1994.
18. Cyrus, M., Beck, J., Generalized two and three dimensional clipping. Computers & Graphics, 1979, 3, 23-28.
19. Skala,V.: An Efficient Algorithm for Line Clipping by Convex and Non-Convex Polyhedrons in E3, Computer Graphics Forum, Vol.15, No.1, pp.61-68, 1996.
20. Skala,V.: A Fast Algorithm for Line Clipping by Convex Polyhedron in E3, Computers & Graphics, Pergamon Press, Vol.21, No.2, pp.209-214, 1997.
21. Skala,V., Kolingerova, I., Blaha, P.: A comparison of 2d line clipping algorithms, Machine Graphics&Vision, Vol.3,No.4, pp.625-633, 1995.
22. Bui,D.H., Skala,V.: New Fast Line Clipping Algorithm in E2 with O(lg N) Complexity, SCCG'99 Int.Conf. proceedings, Budmerice, Slovak Republic, pp.221-228, 1999.